Universidad Técnica Federico Santa María

DEPARTAMENTO DE INGENIERÍA ELECTRONICA

ELO 325 - Seminario de Computadores I

Guía 2

Estudiante **ROL**
Juan Aguilera Castillo 201621028-8

Paralelo: 1

Profesor

Fernando Auat

Ayudante

Nicolás Aguayo

Fecha : 26 de diciembre de 2021

Índice

Índice de figuras

Índice de cuadros

1. Información adicional

Para el desarrollo de esta guía programó usando `Python` y MATLAB, adicionalmente para la reproducción de los códigos presentes en **Anexos** se debe tener en cuenta que los programas *Python* fueron desarrollados en *VS Code*, instalando las siguientes bibliotecas:

- numpy
- matplotlib
- keyboard
- sim

Todos los códigos, el proyecto en `CoppeliaSim` y videos demostrativos se encuentran en este enlace.

2. Sensibilidad del sensor

2.1.

El sensor *LiDar* presentado por el ayudante para realizar las pruebas experimentales corresponde al modelo *UTM-30LX-EW*, cuyo *datasheet* se muestra en este enlace. De este se extraen los parámetros de relevancia, algunos de ellos se pueden ver en la Figura 1. La longitud de onda que utiliza es de 908 nm, el máximo rango de detección corresponde a los 30 metros, su ángulo de detección es de 270 grados, por lo que su zona ciega consta de los 90 grados detrás de este. Además, tiene una resolución angular de 0.25 grados y una velocidad de *scan* de 25ms. En cuanto a la precisión del sensor, esta es de $\mp 30mm$ entre 0.1 a 10 metros, y de $\mp 50mm$ entre 10 a 30 metros, en un ambiente interior.

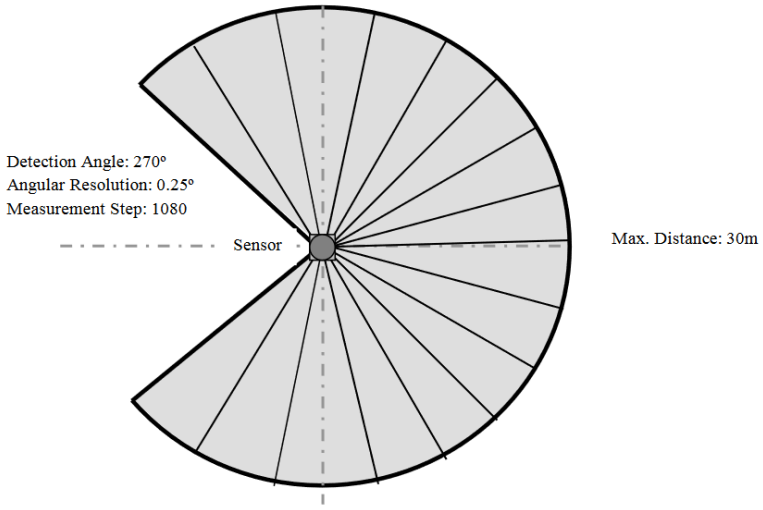

Figura 1: Esquema de sensor *LiDaR*.

2.2.

Los datos utilizados para esta experiencia se encuentran en este repositorio, los cuales corresponden a los *scans* extraídos por parte del ayudante mediante el sensor *LiDar*. Además, el código utilizado para procesar la información se encuentra en este repositorio.

Para extraer el sector donde se encuentra el lado de la caja visto por el sensor se utiliza el siguiente *script* escrito en MATLAB:

```
index = find(abs(x1) < largo_real);
x1 = x1(index(1):index(length(index)));
y1 = y1(index(1):index(length(index)));
x = x1;
[y1,removed] = rmoutliers(y1,'median');
for i = 1:length(removed)
    if(removed(i))
        x(x==x1(i))=[];
    end
end
[x,removed] = rmoutliers(x,'median');
```

```
y = y1;
for i = 1:length(removed)
    if(removed(i))
        y(y==y1(i))=[];
    end
end
```

Donde se almacenan los datos adquiridos por el sensor para cada *scan* en **x1** e **y1**. Los datos en **x1** buscan representar el largo de la caja, por lo que se acotan sus posibles valores, luego se usa la función *rmoutliers* usando el método de los *median* para guardar los índices más cercanos entre sí. Se realiza algo similar con la variable **y1**Se hicieron pruebas con distintos métodos de esta función y este arrojo el menor error..

Para demostrar la extracción de datos del código, en la Figura 2 se aprecian los datos extraídos por el sensor en un *scan*, en la Figura 3 se muestra los datos que representan el frente de la caja según el *script* recién descrito.

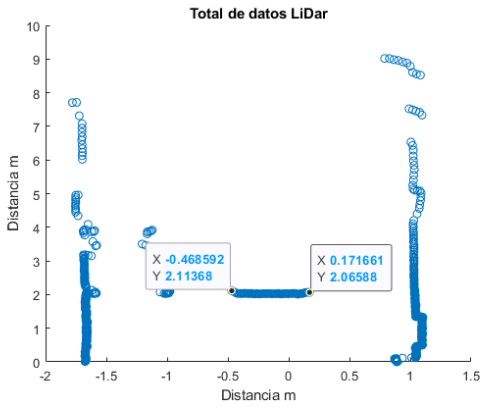

Figura 2: Datos extraídos por *LiDaR*.

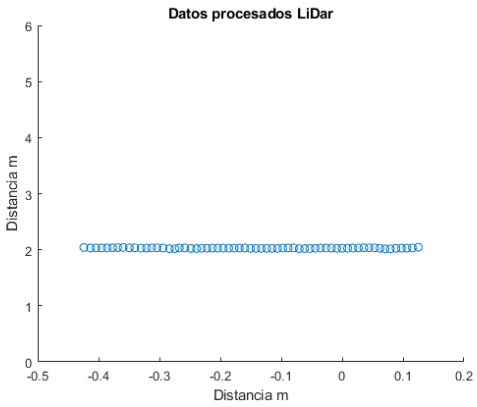

Figura 3: Datos de *LiDaR* procesados.

Con los valores extraídos se calcula la recta que pasa por estos puntos usando mínimos cuadrados, esto se realiza con el siguiente *script*:

```
[p1, p2] = polyfit(x,y,1);
error_medicion = 0;
for z = 1:length(x)
    error = abs(y(z)-(x(z)*p1(1)+p1(2)));
    error_medicion = error_medicion + error;
end
error_medicion = error_medicion/length(x);
```

Donde la función *polyfit* se encarga de encontrar la recta que pasa por los puntos **x, y** encontrados anteriormente con el menor error. Luego se calcula el error, que se define como el promedio de la distancia entre cada punto a la recta calculada.

Se repitieron las mediciones del error para distintas distancias y distintas cajas. En la Figura 4 se muestran los resultados obtenidos.

Respecto a estos errores, sabiendo que la longitud de onda de la señal utilizada por el sensor es de 908 nm y tomando en cuenta las longitudes de onda de los colores visibles, como se muestra en la Figura 5, observamos que la ausencia de color (negro) tiene un mayor error para todas las distancias, siendo este el

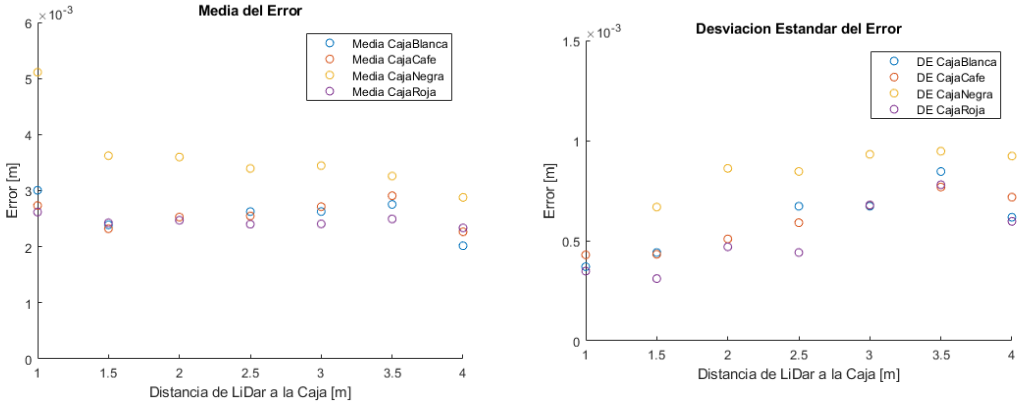

Figura 4: Media y Desviación estándar.

menos **exacto**, debido a que este no refleja tanto la señal, para los otros colores, más cercanos a los 908 nm, tienen un menor error, en especial el rojo, por ser el más cercano (El blanco al refleja todas las longitudes de onda es igualmente **exacto**). Al analizar las desviaciones estándar, notamos que esta aumenta con la distancia, por lo que decimos que el sensor es más **preciso** mientras más cerca se encuentra, para todos los colores (también el color rojo demuestra las detenciones más precisas).

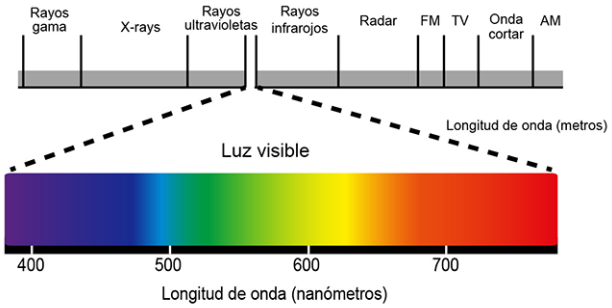

Figura 5: Longitud de onda de los colores visibles.

3. Evasión de obstáculos

El programa completo escrito en *Python* para este ejercicio se ecuentra en este repositorio.

Se programó una función que, a partir de la posición global del robot y de la posición global del objetivo a alcanzar, genere una secuencia de puntos que los una, así como el ángulo del robot necesario para alcanzar cada punto, esta se muestra a continuación:

```
def rect_generator(pos_in, pos_goal, H):
    tolerancia = H
    xi, yi,_ = pos_in
    xf, yf,_ = pos_goal

    alpha = math.atan2((yf-yi),(xf-xi))

    xdata = []
    ydata = []
    thdata = []

    x0 = xi # x_{k}
    y0 = yi # y_{k}

    while(1):

        x = math.cos(alpha)*H + x0
        y = math.sin(alpha)*H + y0

        if((y-y0) != 0 and (x-x0) != 0):
            th = math.atan2((y-y0),(x-x0))
        else:
            th = 0

        xdata.append(x)
        ydata.append(y)
        thdata.append(th)

        x0 = x
        y0 = y
        if( (np.linalg.norm(np.array((x,y))-np.array((xf,yf)))
    < 2*H)):
            break
        elif( ( (x >= xf-tolerancia and x <= xf+tolerancia)
   and (y >= yf-tolerancia and y <= yf+tolerancia) ) ):
            break
    xdata.append(xf)
    ydata.append(yf)
    thdata.append(th)
    return xdata, ydata, thdata
```

Se programó una función que, a partir de los puntos generados por la función anterior, es capaz de entregar las velocidades y ángulos de la rueda necesarios para alcanzarlos. En esta función se utilizó el modelo cinemático del triciclo, ya que el modelo de *car-like* utilizado necesita una velocidad y un ángulo para moverse. Esta función se muestra a continuación:

```
def calculate_vs (x0,y0,th0,x,y,th,dT):
```

```
    d = 0.6 # distancia entre las ruedas [m]

    alpha = 0
    V = 0

    if(math.cos(th0) != 0 ):
        Vcos = (x-x0)/(dT*math.cos(th0))
    else:
        Vcos = float("inf")
    Vsin = d*(th-th0)/dT

    alpha = math.atan2(Vsin,Vcos)
    if(math.cos(alpha) != 0):
        V = Vcos/math.cos(alpha)
    else:
        V = float("inf")
    return V, alpha
```

Para detectar los obstáculos tomando el vector de datos que entrega el sensor *fastHokuyo* se implementó una función que recibe la posición actual del robot, su ángulo y los datos del sensor, y retorna la posición global del obstáculo detectado, esta función se muestra a continuación:

```
def sense_obstacles(pos, carAngle, lidarData):
    xdata=[]
    ydata=[]
    i = 0
    x1, y1,_ = pos
    for angle in np.linspace(0,(4/3)*math.pi,len(lidarData),
    False):
        if(lidarData[i] > 0 ):
            x2, y2 = (x1 + lidarData[i]*math.cos(angle+(
    carAngle-(4/3)*math.pi/2)), y1 + lidarData[i]*math.sin(
    angle+(carAngle-(4/3)*math.pi/2)))
            xdata.append(x2)
            ydata.append(y2)
        i += 1
    return xdata, ydata
```

La idea del algoritmo de *bug* 1 a implementar es:

- Trazar el camino de puntos que conecta la posición inicial con el objetivo.
- Seguir la línea.
- Si se encuentra un obstáculo de frente, el robot gira a la derecha.
- Si el obstáculo esta de lado del robot, este lo rodea hasta encontrase nuevamente con el camino de puntos generado en un principio.
- Una vez de vuelta en seguir la línea, se repute el algoritmo hasta llegar a la meta.

Los resultados de esta implementación se muestran en el video disponible en este enlace.

4. Anexos